

```

006 class Parent
007 {
008 public:
009     void M1()
010     {
011         cout << "Parent::M1" << endl;
012     }
013     virtual void M2()
014     {
015         cout << "Parent::M2" << endl;
016     }
017     void M3()
018     {
019         cout << "Parent::M3" << endl;
020         M4();
021     }
022     virtual void M4()
023     {
024         cout << "Parent::M4" << endl;
025     }
026 };
027
028 class Child : public Parent
029 {
030 public:
031     void M1()
032     {
033         cout << "Child::M1" << endl;
034     }
035     virtual void M2()
036     {
037         cout << "Child::M2" << endl;
038     }
039     virtual void M4()
040     {
041         cout << "Child::M4" << endl;
042     }
043 };
044
045 void Caller1( Parent *p )
046 {

```

```

047     p->M1();
048     p->M2();
049 }
050
051 void Caller2( Parent *p )
052 {
053     p->M3();
054 }
055
056 void TestVirtual()
057 {
058     Parent p;
059     Caller1( &p ); // Parent::M1\nParent::M2
060     Child c;
061     Caller1( &c ); // Parent::M1\nChild::M2
062
063     Caller2( &p );
064     Caller2( &c );
065 }
066
067 typedef Parent * PParent;
068 void TestMemory()
069 {
070     {
071         int *p = new int;
072         *p = 100;
073         cout << *p << endl;
074         delete p;
075         p = 0;
076
077         Parent *base = new Child();
078         base->M1();
079         base->M2();
080         delete base;
081         base = 0;
082     }
083     {
084         int *p = new int[10];
085         for( int i = 0; i < 10; ++i )
086         {
087             p[i] = i * i;

```

```

088     }
089     delete[] p;
090     p = 0;
091
092     // Create using default constructor
093     Parent *bases = new Parent[5];
094     for( int i = 0; i < 5; ++i )
095     {
096         bases[i].M1();
097         bases[i].M2();
098     }
099     delete[] bases;
100     bases = 0;
101
102     // 5 pointers to Parent, no memory allocation
103     // Parent *bases2[5];
104     PParent bases2[5];
105     cout << "Virtual test" << endl;
106     for( int i = 0; i < 5; ++i )
107     {
108         if ( ( i % 2 ) == 0 )
109         {
110             bases2[i] = new Parent();
111         }
112         else
113         {
114             bases2[i] = new Child();
115         }
116         bases2[i]->M1();
117         bases2[i]->M2();
118     }
119
120     for( int i = 0; i < 5; ++i )
121     {
122         delete bases2[i];
123         bases2[i] = 0;
124     }
125 }
126 }
127
128 class ClassA

```

```

129 {
130 public:
131     ClassA()
132     {
133         cout << "ClassA::ClassA" << endl;
134     }
135     virtual ~ClassA()
136     {
137         cout << "ClassA::~ClassA" << endl;
138     }
139 };
140 };
141
142 class ClassB : public ClassA
143 {
144 public:
145     int *p;
146     ClassB() :
147         ClassA()
148     {
149         p = new int[10];
150         cout << "ClassB::ClassB, new 10 ints" << endl;
151     }
152     virtual ~ClassB()
153     {
154         delete[] p;
155         cout << "ClassB::~ClassB, free 10 ints" << endl;
156     }
157 };
158 };
159
160 void TestDestructors()
161 {
162     {
163         ClassA a;
164         ClassB b;
165     }
166     {
167         ClassA *base = new ClassB();
168         // If destructor is not virtual then
169         // ~ClassA will be called and memory is not freed

```

```

170     delete base;
171 }
172 }
173
174 void ChangeIntPtr( int *x )
175 {
176     *x = 100;
177 }
178
179 void ChangeIntRef( int &x )
180 {
181     x = 200;
182 }
183
184 void Caller3( Parent &p )
185 {
186     p.M1();
187     p.M2();
188 }
189
190 void TestReferences()
191 {
192     int x;
193     ChangeIntPtr( &x );
194     cout << x << endl;
195     ChangeIntRef( x );
196     cout << x << endl;
197
198     int *pX = &x;
199     *pX = 300;
200     cout << x << " " << *pX << endl;
202     int &refX = x;
203     refX = 400;
204     cout << x << " " << refX << endl;
205

```

```

206     Parent p;
207     Child c;
208     Caller3( p );
209     Caller3( c );
210     Parent *p2 = new Child();
211     Caller3( *p2 );
212     delete p2;
213     p2 = 0;
214     //Caller3( *p2 ); // runtime error
215 }

```

```

009 const int FPS = 60;
010 const int SCREEN_W = 640;
011 const int SCREEN_H = 480;
012
013 class Figure
014 {
015 protected:
016     double x_;
017     double y_;
018     double dx_;
019     double dy_;
020
021 public:
022     Figure()
023     {
024         Reset();
025     }
026
027     void Reset()
028     {
029         x_ = rand() % SCREEN_W;
030         y_ = rand() % SCREEN_H;
031         dx_ = 10.0 - rand() % 21;
032         dy_ = 10.0 - rand() % 21;
033     }
034
035     virtual void Draw(){}
036
037     virtual void Move()
038     {
039         x_ += dx_;
040         y_ += dy_;
041         if ( ( x_ < 1.0 ) || ( x_ > SCREEN_W ) ||
043             ( y_ < 1.0 ) || ( y_ > SCREEN_H ) )
044         {
045             Reset();
046         }
047     }
048 };
049
050 typedef Figure * PFigure;
051
052 class Square : public Figure
053 {
054 protected:
055     double a_;
056 public:
057     Square( double a ) :
058         Figure(),
059         a_( a )
060     {
061     }
062     virtual void Draw()
063     {
064         double half = a_ / 2;
065         al_draw_filled_rectangle( x_ - half, y_ - half,
066             x_ + half, y_ + half, al_map_rgb( 255, 0, 0 ) );
067     }
068 };
069
070 class Circle : public Figure
071 {
072 protected:
073     double r_;
074     unsigned char color_;
075 public:
076     Circle( double r ) :
077         Figure(),
078         r_( r ),
079         color_( rand() % 256 )
080     {
081     }
082     virtual void Draw()
083     {
084         ++color_;
085         al_draw_filled_circle( x_, y_, r_, al_map_rgb( 0,
086             color_, 0 ) );
087     }
088
089     const int MAX = 100;
090
091     class ScreenSaver
092     {

```

```

092 private:
093     PFigure figures[MAX];
094     int size_;
095
096 public:
097     ScreenSaver() :
098         size_( 0 )
099     {
100         // Set to null all pointers
101         memset( figures, 0, sizeof( figures ) );
102     }
103
104     ~ScreenSaver()
105     {
106         for( int i = 0; i < size_; ++i )
107         {
108             delete figures[i];
109             figures[i] = 0;
110         }
111     }
112
113     void Draw()
114     {
115         al_clear_to_color( al_map_rgb( 0, 0, 0 ) );
116         for( int i = 0; i < size_; ++i )
117         {
118             figures[i]->Draw();
119         }
120     }
121
122     void Next()
123     {
124         for( int i = 0; i < size_; ++i )
125         {
126             figures[i]->Move();
127         }
128     }
129
130     void Add( Figure *f )
131     {
132         if ( size_ >= MAX )
133         {
134             return;

```

```

135         }
136         figures[ size_ ] = f;
137         ++size_;
138     }
139
140 };
141
142 ScreenSaver ss;
143
144 void fps()
145 {
146     ss.Next();
147 }
148
149 void draw()
150 {
151     ss.Draw();
152 }
153
154 int main(int argc, char **argv)
155 {
156     srand( time(0) );
157     if( !InitAllegro( SCREEN_W, SCREEN_H, FPS ) )
158     {
159         DestroyAllegro();
160         return 1;
161     }
162     for( int i = 0; i < 100; ++i )
163     {
164         if ( ( i % 2 ) == 0 )
165         {
166             ss.Add( new Circle( 10.0 + rand() % 30 ) );
167         }
168         else
169         {
170             ss.Add( new Square( 10.0 + rand() % 30 ) );
171         }
172     }
173     RunAllegro( &fps, &draw );
174     DestroyAllegro();
175     return 0;
176 }

```